Computing Homology: A Global Reduction Approach

David Corriveau[1] and Madjid Allili[2]

[1] Dept. of Computer Science, Université de Sherbrooke, Sherbrooke, QC, J1K2R1
david.corriveau@usherbrooke.ca
[2] Dept. of Computer Science, Bishop's University, Sherbrooke, QC, J1M0C8
mallili@ubishops.ca

Abstract. A new algorithm to compute the homology of a finitely generated chain complex is proposed in this work. It is based on grouping algebraic reductions of the complex into structures that can be encoded as directed acyclic graphs. This leads to sequences of projection maps that reduce the number of generators in the complex while preserving its homology. This organization of reduction pairs allows to update the boundary information in a single step for a whole set of reductions which shows impressive gains in computational performance compared to existing methods. In addition, this method gives the homology generators for a small additional cost.

Keywords: Homology Computation, Reduction, Directed Acyclic Graphs, Generators.

1 Introduction

Computation of homology has become a very important tool in many applications and domains such as dynamical systems, image processing and recognition, and visualization. In dynamics, invariants such as the Conley index are computed using homology algorithms. In digital image analysis, topological invariants are useful in shape description, indexation, and classification. Among shape descriptors based on homology theory, there is the Morse shape descriptor [1,2], and the persistence barcodes for shape [3]. The necessity of improved algorithms appears evident as new applications of the homology computation arise in research for very large data sets. The classical approach to compute homology of a chain complex with integer coefficients reduces to the calculation of the Smith Normal Form (SNF) of the boundary matrices which are in general sparse [4,5]. Unfortunately, this approach has a very poor running-time and its direct implementation yields exponential bounds. Uses and improvements of this method can be found in works that appeared recently [6,7]. Another generic approach is the method of reduction proposed in [8,5]. Several algorithms based on the idea of reduction and that improve the time complexity for particular types of data sets have been designed. For cubical sets embedded in $\mathbb{R}^3$, Mrozek et al.

S. Brlek, C. Reutenauer, and X. Provençal (Eds.): DGCI 2009, LNCS 5810, pp. 313–324, 2009.

proposed a method [9] based on the computation of acyclic subspaces by using lookup tables. For problems dealing with higher dimensions, Mrozek and Batko developed an algorithm [10] based on the concept of coreduction which showed interesting performance results on cubical sets. However problems in which data is higher dimensional and not cubical are still difficult to handle. The classical methods spend most of their time in updating the boundary homomorphisms after each reduction step.

We propose a method that identifies reduction pairs and organizes them in a structure that allows to efficiently update the boundary information in a single step for a whole set of reductions. This reduces to the minimum the manipulations of the data structures that store the boundary information. Starting at an arbitrary cell A with a face a in its boundary, we identify successive adjacent admissible pairs of reduction and build the longest possible sequence originating at A. Several sequences can originate at the same cell, and each sequence can be seen as a path in a directed acyclic graph, called a *reduction DAG*, where nodes are reduction pairs and oriented edges denote an adjacency relation between the reduction pairs. Given a reduction DAG across a chain complex, we achieve a global simplification of the complex by performing all the reductions in the DAG at once. We will establish direct algebraic formulas that allow to update the boundary of remaining cells. The net advantage of our approach is that the boundaries are not explicitly updated after each reduction step. Instead, this information is always preserved implicitly within the data structures and processed globally for each reduction DAG allowing to reduce considerably the computational time.

2 Chain Complexes and Homology

A finitely generated free *chain complex* $(\mathcal{C}, \partial)$ with coefficients in a ring $\mathcal{R}$ is a sequence of finitely generated free abelian groups $\{C_q\}_{q\in\mathbb{Z}}$ together with a sequence of homomorphisms, called boundary maps or operators, $\{\partial_q : C_q \to C_{q-1}\}_{q\in\mathbb{Z}}$ satisfying $\partial_{q-1} \circ \partial_q = 0$ for all $q \in \mathbb{Z}$. Typically, $C_q = 0$ for $q < 0$. For each p, the elements of C_p are called p-chains and the kernel of $\partial_p : C_p \to C_{p-1}$ is called the group of p-cycles and denoted $Z_p = \{c \in C_p \mid \partial_p c = 0\}$. The image of ∂_{p+1}, called the group of boundaries, is denoted by $B_p = \{c \in C_p \mid \exists\, b \in C_{p+1}$ such that $\partial_{p+1} b = c\}$. B_p is a subgroup of Z_p because of the property $\partial_p \circ \partial_{p+1} = 0$. The quotient groups $H_p := Z_p/B_p$ are the homology groups of the chain complex $\mathcal{C}$. The reduction of a chain complex is a procedure that consists of removing successively pairs of generators from the bases of its chain groups while preserving the homology of the original complex. At the algebraic level, each removal of a pair of generators that form a reduction pair is equivalent to a collection of projection maps $\{\pi_d : C_d \to C_d\}_d$ that send each generator of the removed pair into 0. Moreover, it projects the other cells into the remaining generators taking into account the modifications of their boundaries caused

by the removal of the pair. Let $C'_d = \pi_d(C_d)$ for each d. It is shown in [5] that C' is a chain complex and $H_*(C') \cong H_*(C)$, that is, the homologies of C' and C coincide. To calculate the homology of the original chain complex C, the idea is to define a sequence of projections associated to the removal of pairs in all dimensions and then compute the homology of the resulting chain complex that is the image of the successive projections. A sequence of projections is complete if no other projection can be added to the sequence. Such a sequence always exits when the chain complex is finitely generated. Let (C, ∂) be the original chain complex and (C^f, ∂^f) is the final chain complex obtained after a complete sequence of projections. We already know that the two complexes have the same homology, moreover it is easily observed that if $\partial^f = 0$ then $H_*(C) \cong H_*(C^f) \cong C^f$, that is, the Betti number β_d is given by the number of d-cells remaining in the complex C^f.

Formally, a reduction pair and its associated collection of projection maps are defined as follows.

Definition 1. *Let $\mathcal{C} = \{C_d, \partial_d\}$ be an abstract chain complex. A pair of generators (a, B) such that $a \in C_{m-1}$, $B \in C_m$ and $\langle \partial B,\ a \rangle = \pm 1$ is called a* reduction pair*. It induces a collection of group homomorphisms*

$$\pi_d c := \begin{cases} c - \frac{\langle c,\ a \rangle}{\langle \partial B,\ a \rangle} \partial B & \text{if } d = m-1, \\ c - \frac{\langle \partial c,\ a \rangle}{\langle \partial B,\ a \rangle} B & \text{if } d = m, \\ c & \text{otherwise}, \end{cases}$$

where $c \in C_d$ and $\langle \cdot\ ,\ \cdot \rangle$ is the canonical bilinear form on chains.

3 Computation of Homology by Grouping Reductions

Forming RDAGs: For a fixed dimension and starting from an arbitrary cell, we form the maximal possible directed acyclic graph (DAG) whose nodes are reduction pairs and a directed edge between two pairs (a, B) and (c, D) means that (a, B) is adjacent to (c, D) (see Section 3.1). A directed path in the DAG corresponds to a reduction sequence. That type of reduction sequence affects only adjacent cells to the path and leaves other cells unchanged. Information about each reduction is carried out by the reduced cells. After performing the whole set of reductions, we obtain the reduced complex by extracting the boundary information from the data structures associated to the cells visited by the reduction DAGs. We introduce the main concepts through an example. The example details how the projections of cells are calculated given a reduction sequence. This will show what information is exactly needed to be kept about the original complex in order to be able to rebuild the reduced complex.

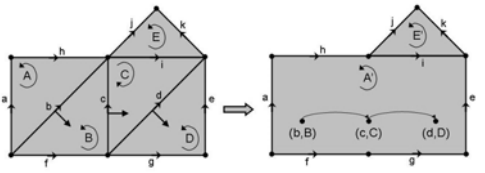


Fig. 1. Example of collapsing by using RDAGs

Example 1. Consider the complex given in Figure 1. The depicted sequence of reduction pairs originating at the cell A is $(b, B), (c, C)$ and (d, D). The projection maps at the level of the 2-cells give the following

$$(b,B): \quad \pi_2^b \alpha = \alpha - \frac{\langle \partial\alpha,\ b\rangle}{\langle \partial B,\ b\rangle} B = \alpha - \lambda_b B$$
$$(c,C): \quad \pi_2^c \alpha = \alpha - \frac{\langle \partial\alpha,\ c\rangle}{\langle \partial C,\ c\rangle} C = \alpha - \lambda_c C$$
$$(d,D): \quad \pi_2^d \alpha = \alpha - \frac{\langle \partial\alpha,\ d\rangle}{\langle \partial D,\ d\rangle} D = \alpha - \lambda_d D$$

It follows that the projection map associated to the whole sequence is $\pi_2 = \pi_2^d \circ \pi_2^c \circ \pi_2^b$. The 2-cell A is the only cell adjacent to the sequence while the 2-cell E is not adjacent. Their respective projections with respect to the sequence are

$$\pi_2 A = \pi_2^d \circ \pi_2^c (A - \lambda_b B) = A - \lambda_b B + \lambda_b \lambda_c C - \lambda_b \lambda_c \lambda_d D.$$
$$\pi_2 E = E' = \pi_2^d \circ \pi_2^c \left(E - \frac{\langle \partial E,\ b\rangle}{\langle \partial B,\ b\rangle} B \right) = \pi_2^d \circ \pi_2^c (E - 0 \cdot B) = \ldots = E.$$

Each coefficient in the projection is called the coefficient of contribution of the corresponding cell to the projection of A.

3.1 Projection Formulas for Grouped Reductions

We define how to organize the reductions and the complex using reduction DAGs. A cell A_1 is *adjacent* to the pair (a_2, A_2) if the cells A_1 and A_2 are of the same dimension m and a_2 is a cell with dimension $m-1$ in the boundaries of both A_1 and A_2. A pair (a_1, A_1) is *adjacent* to a pair (a_2, A_2) if A_1 is adjacent to (a_2, A_2). A *reduction DAG* is a directed acyclic graph whose nodes are reduction pairs and a directed edge between two pairs (a_1, A_1) and (a_2, A_2) means that (a_1, A_1) is adjacent to (a_2, A_2).

A *path* P from (a_1, A_1) to (a_n, A_n) in a reduction DAG G is a *reduction sequence* $(a_1, A_1), \ldots, (a_n, A_n)$ whose elements are nodes in G and (a_i, A_i) is adjacent to (a_{i+1}, A_{i+1}), for $i = 1...n-1$. A cell A is said to be *adjacent* to a reduction sequence if it is adjacent to some pair of the sequence. A path from

(a_1, A_1) to (a_n, A_n) in a reduction DAG G is said to *originate at* A_0 if A_0 is adjacent to (a_1, A_1). A cell not appearing in any reduction DAG is called a *projected* cell. The cells that appear in a reduction DAG are called *reduced* cells.

In the following theorem, we give a formula to calculate the projection of a cell by considering the contribution of a single path and ignoring the input from other paths and sub-paths. This formula is called "path projection" and is proved using induction on the length of the sequence of reductions.

Theorem 1. *Let* $P : (a_1, A_1), \ldots, (a_n, A_n)$ *be a path originating at* A_0. *The* path projection of A_0 *by the path* P *is given by* $\pi_P(A_0) = A_0 + \sum_{i=1}^{n} \Gamma_i A_i$, *where* $\Gamma_i = (-1)^i \prod_{j=1}^{i} \lambda_j$, *and* $\lambda_j = \frac{\langle \partial A_{j-1},\ a_j \rangle}{\langle \partial A_j,\ a_j \rangle}$ *for* $1 \leq i, j \leq n$. Γ_i *is called the* coefficient of contribution *of* A_i *in the path projection of* A_0.

Proof: The proof is easily done by induction on the length of the path. □

Now, we can write $\pi_{P_n}(A_0) = A_0 + \underbrace{\sum_{i=1}^{n} \Gamma_i A_i}_{\Psi_{P_n}}$. We denote by Ψ_{P_n} the *projection chain* of the cell A_0 by the path P_n.

Corollary 1. *Let* $P_1, P_2, \ldots, P_k$ *be disjoint non overlapping paths originating at* A_0. *The total projection of* A_0 *by* $P_1, \ldots, P_k$ *denoted by* $\pi_{P_1, P_2, \ldots, P_k}(A_0) = A_0 + \sum_{i=1}^{k} \Psi_{P_i}$ *where* Ψ_{P_i} *is the* projection chain *of the cell* A_0 *by the path* P_i.

Corollary 2. *Let* T *be a reduction tree originating at* A_0, *then the projection of* A_0 *by* T *is equal to the sum of* A_0 *and the* projection chain *of* A_0 *by* T.

Proof: Typically, the trees originating at A_0 can occur as pure paths, in which case the associated projections are given previously. Otherwise, we can find paths that share a common ancestral branch that originates at A_0. This is seen as a bifurcation as shown in Figure 2. In this case, the ancestral branch is a path $B : (b_1, B_1), \ldots, (b_{n_B}, B_{n_B})$ which is extended by two paths $C : (c_1, C_1), \ldots, (c_{n_C}, C_{n_C})$ and $D : (d_1, D_1), \ldots, (d_{n_D}, D_{n_D})$. The combination of

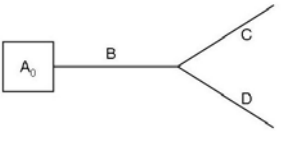


Fig. 2. A path B bifurcates into two paths C and D

the results in Theorem 1 and Corollary 1 can be used to show that the total projection of A_0 by $T = (BC), (BD)$ is given by

$$\pi_T(A_0) = A_0 + \sum_{i=1}^{n_B} \Gamma_i B_i + \Gamma_{n_B} \left(\sum_{j=1}^{n_C} \alpha_j C_j + \sum_{k=1}^{n_D} \beta_k D_k \right)$$

The term $\sum_{i=1}^{n_B} \Gamma_i B_i + \Gamma_{n_B} \left(\sum_{j=1}^{n_C} \alpha_j C_j + \sum_{k=1}^{n_D} \beta_k D_k \right) = \pi_T(A_0) - A_0 = \Psi_T(A_0)$ is called the *projection chain of the tree.* □

Theorem 2. *Let A_0 be a cell adjacent to a RDAG in a chain complex C. The projection of A_0 by the RDAG is equal to A_0 to which we add the projection chains of A_0 by all the trees in the RDAG that originate at A_0, that is*

$$\pi_{RDAG}(A_0) = A_0 + \sum_{T \in \mathcal{T}_{A_0}} \Psi_T(A_0)$$

where $\mathcal{T}_{A_0}$ is the collection of all reduction trees in the RDAG originating at A_0.

Proof: The results proved for paths in corollaries 1 and 2 can be easily extended for the case of trees to find the formula for the projection of A_0.

Building a Simplified Complex: Using reduction DAGs to compute the homology of a chain complex is a recursive process. At each recursion level, the algorithm simplifies the complex by constructing reduction DAGs on the complex and saving the associated projections into appropriate data structures. This is performed simultaneously for each dimension. This process eventually stops when it is impossible to add another reduction pair to any reduction DAG. In that case, the algorithm will build the associated simplified complex and continue the reduction process on the simplified complex. The simplified complex is rebuilt from the projected cells only (reduced cells are not considered). The boundaries of the cells are updated using global projection formulas that allow to calculate the incidence numbers between cells of contiguous dimensions. Note that the reduced cells are not completely removed from the structures since they may be needed to recover homology generators expressed in terms of cells of the original complex as we explain in subsection 3.2. Contrary to the classical case where the boundary updating is done at each reduction step and may concern cells that can be reduced at a later step, a major benefit of this new approach is that the boundary updating is done only among the projected cells which often constitute a small fraction of the number of cells in the original complex.

3.2 Calculating Generators

Let us consider the complex of a plane quotiented by its boundary as illustrated in Figure 3. This complex is homeomorphic to a 2-sphere and the 2-cell A

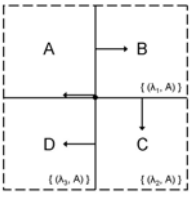


Fig. 3. A Complex homeomorphic to a 2-sphere

represents the 2-generator $A' = A + \lambda_1 B + \lambda_2 C + \lambda_3 D$. As illustrated in Figure 3, after each reduction, the projection coefficients λ_i are saved into the projection lists maintained in each reduced cell. Thus, to get the 2-generator associated to the 2-cell A, one has to scan through each reduced 2-cell and extract the projection coefficients associated to A. Projections correspond to generators but they can be expressed with cells of the simplified complex at any level of the recursion. However, in order for the generators to carry geometric meaning, it makes sense only to express them with cells from the original complex. Due to the recursive simplifications, a projected cell at a previous level of the recursion may become a reduced cell at a later level of the recursion. We illustrate this in Figure 4. In this example, there are two levels of recursion. The final simpli-

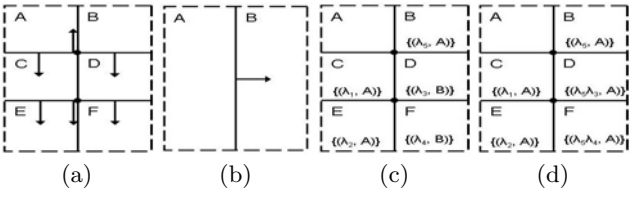
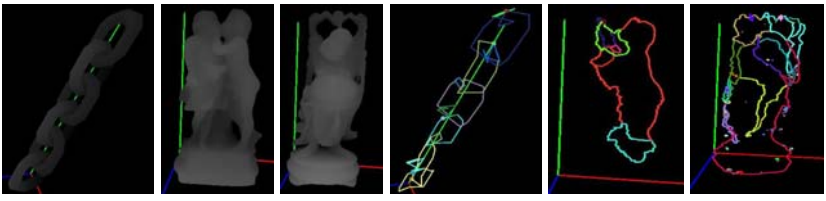


Fig. 4. (a) Reduction 1 of the initial complex. (b) Reduction 2 of the simplified complex. (c) Initial complex after returning from the second reduction. (d) Initial complex after returning from the first reduction.

fied complex is shown in 4(b). At this step, the cell A represents a 2-generator that is expressed as $A'' = A' + \lambda_5 B'$. Returning from the last recursion, we know that $A' = A + \lambda_1 C + \lambda_2 E$ and $B' = B + \lambda_3 D + \lambda_4 F$. The cell A represents a projected cell at both recursion levels (4(b) and 4(c)) and requires no special consideration. On the other hand, the cell B is a projected cell at the first level of recursion but becomes a reduced cell at the second recursion level. Consequently, returning from the second recursion level, we scan through each 2-cell and whenever B is encountered in one of the projection lists, it is replaced by $\lambda_5 A$. This is shown in Figure 4(d). Finally, the generator is expressed by $A'' = A + \lambda_5 B + \lambda_1 C + \lambda_5 \lambda_3 D + \lambda_2 E + \lambda_5 \lambda_4 F$. In Figure 5 we show different 1-generators that we extracted from 3D models.



(a) (b) (c) (d) (e) (f)

Fig. 5. (a,b,c) The 3D models of a chain, two kissing children and a Buddha. (d,e,f) The calculated holes (one dimensional generators).

4 Data Structures and Algorithms

The first data structure represents a chain complex.

ChainComplex:

E : two dimensional array of cells organized by their respective dimension, that is E[d][i] denotes the i-th d-cell.

In our implementation, the pointer to a cell is used to identify the cell. These identifiers are saved in **E**. Only the pointers of the projected cells are copied into the simplified complex.

Cell:

boundary/coboundary: list of faces/cofaces.

state: a flag taking one of the values in {NORMAL, REDUCED, PROJECTED, VISITED}.

projCells: list to save the PROJECTED cells and their associated coefficients that project onto the given cell when it is REDUCED.

nbUpdates: approximates the number of updates to a **projCells** list when the given cell is reduced by one of its cofaces.

Initially, all cells are set to the NORMAL state. REDUCED is assigned to the cells in a reduction sequence and PROJECTED is assigned to the cells adjacent to a reduction sequence. The VISITED flag is assigned to the d-cells that don't have any $(d+1)$ coface that can form an admissible reduction pair. The flag helps to avoid testing more than once if the given d-cells have an admissible reduction pair. The **projCells** list is used by the REDUCED cells to keep track of the coefficients of contribution of every PROJECTED cell for which it contributes. To build a sequence of reduction pairs, **nbUpdates** is used to select reduction pairs that should minimize the amount of updates to the **projCells** list. We now give the principal steps of the algorithm.

$\diamond$ *HOM_RDAG*(**ChainComplex** K) returns the Betti numbers of a chain complex K by using reduction DAGs.

1. (*Initialization*) For all cells in K, set its **state** to NORMAL, empty **projCells**, set **nbUpdates** to the number of cofaces.
2. (*Reduce cells*) Proceed by decreasing order of dimension. Order the d-cells by increasing value of **nbUpdates**. Following this order, start a reduction DAG (call *BuildRDAG*()) from each cell whose state is NORMAL.
   For dimension 0, change remaining NORMAL 0-cells to PROJECTED.
3. (*Build the simplified complex*) Call *BuildSimplifiedComplex*(K). This method extracts the boundary information from the data structures to build a simplified complex K'.
   Continue to recursively simplify K' by calling *HOM_RDAG*(K') until there are no reduction pair left.
   Test if all boundaries are trivial. If not, then continue the reduction process with another method such as SNF.
4. (*Return the Betti numbers*) Assign the number of PROJECTED d-cells to β_d.

$\diamond$ *BuildRDAG*(**Cell** c) builds a reduction DAG from cell c.

1. (*Initialization*) Save c.**nbUpdates** into *nbUpdatesLimit* for later use.
2. (*Find a reduction pair*) Find a coface B of c such that B is NORMAL or VISITED. If a coface B is found, then call *PairCells*(c,B), otherwise set c to VISITED and exit *BuildRDAG*().
3. (*Expand the reduction DAG*) Expand the reduction DAG (repeat step 2) from all NORMAL faces of B that have **nbUpdates** $\leq$ *nbUpdatesLimit*. Proceed in a breadth first search approach.

$\diamond$ *PairCells*(**Cell** c, **Cell** B) adds the reduction pair (c, B) to the sequence and updates the data structures accordingly.

1. (*Update* ***projCells***) For all cofaces C of c different than B, if C is REDUCED, then add $\lambda_c * C$.**projCells** to B.**projCells**. Otherwise, set C to PROJECTED and add $\lambda_c * C$ to B.**projCells**, where $\lambda_c = -\frac{\langle \partial C,\ c\rangle}{\langle \partial B,\ c\rangle}$.
2. (*Update the* ***nbUpdates*** *variable*) For all faces a of B different than c, add $|B.\textbf{projCells}| - 1$ to c.**nbUpdates**. For all faces of c, remove one to **nbUpdates**.

$\diamond$ *BuildSimplifiedComplex*(**Complex** K) extracts the boundary and coboundary information from the data structures to build a simplified complex K'.

1. (*Update* ***boundary***) Proceed by increasing order of dimension. Let c be a PROJECTED d-cell of K. For all cofaces C of c, iterate through C.**projCells**. Let B be a $(d+1)$-cell in C.**projCells** and λ_B its associated coefficient. Add $\lambda_B \langle \partial C,\ c\rangle\, c$ to B.**boundary** and update c.**coboundary** accordingly.
   Finally, remove all REDUCED cells remaining in the **boundary** and **coboundary** of PROJECTED cells. Copy the pointers of all PROJECTED cells into $K'.E$.

5 Experimental Results and Discussion

We experimented with datasets from d-balls, d-spheres, tori, Bing's houses, 3D medical scans or randomly generated d-complexes. Each dataset contains many files of complexes of various data sizes. The size of a complex is measured as the number of cells plus the number of links (entries in boundary lists). A test is an execution of the algorithm on a file of a specified data set. We performed 30 tests for each file and computed the average, maximum and standard deviation of the time used for calculating the homology, the generators and sorting. We measured the sorting time because it has a time complexity of $\Theta(n \cdot \log n)$ and we wanted to verify that sorting would not monopolize the computation time. Also, we saved statistics on the number of recursions to see its influence on the global performance. We grouped and summarized the results per dataset. Table 1 shows the comparison between the performance of the RDAGs algorithm and the classical reduction method on different datasets. We can observe a significant improvement of the new method. The results suggest subquadratic time complexity for the RDAGs algorithm which is what we measured in Table 2.

Table 1. Reduction DAGs algorithm versus the classical reduction method

Dataset	Data Size	Times Faster
3-Ball	115905	195.4
4-Ball	27841	147.3
2-Sphere	517145	65.0
3-Sphere	134777	182.5
Torus	418608	36.0
Heart	545492	50.5
3-Complex	265322	410.0
5-Complex	431186	458.8
8-Complex	330285	185.9

The second column (**N**) gives the number of files within the dataset. The third and fourth columns show the parameters α and β that best-fit the equation "time $\approx \beta \cdot \text{data\_size}^{\alpha}$", where time is expressed in seconds. We obtained the best-fits from the times measured on the individual files of each dataset. In the last four columns, we present the times in milliseconds that we get from the best-fit equation for different data sizes. Those times approximate the real times measured in real experiments. In Table 2 we report the approximative time that our algorithm uses to calculate homology on various datasets.

In Table 2, we observe that $\alpha \approx 1.16$ and $\beta \approx 5E-8$ for d-balls and d-spheres without regard of the dimension. Except for few values, α is relatively constant while β gradually decreases as the dimension increases. We explain this by the fact that the ratio of exterior face reductions versus interior face reductions increases with the dimension. We observe that the approximation times and the

Table 2. Approximated performance with respect to dataset, data size and dimension

Dataset	N	Equation		Time (msec) Vs Data Size			
		β	α	10^4	10^5	10^6	10^7
2-Ball	15	9.05E-8	1.11	2.49	32.11	413.66	5329.04
3-Ball	11	4.63E-8	1.15	1.84	26.04	367.77	5194.95
4-Ball	11	4.26E-8	1.14	1.55	21.35	294.72	4068.27
5-Ball	9	5.66E-8	1.12	1.71	22.53	297.04	3915.76
6-Ball	5	3.56E-8	1.15	1.42	20.02	282.78	3994.39
2-Sphere	17	5.77E-8	1.18	3.03	45.83	693.71	10499.67
3-Sphere	16	2.79E-8	1.22	2.12	35.12	582.91	9673.96
4-Sphere	10	3.38E-8	1.17	1.62	23.93	353.93	5235.00
5-Sphere	9	5.52E-8	1.14	2.00	27.67	381.89	5271.56
6-Sphere	5	3.30E-8	1.17	1.58	23.36	345.55	5111.09
7-Sphere	3	1.31E-8	1.24	1.19	20.76	360.80	6270.05
Torus	5	1.48E-7	1.07	2.82	33.13	389.28	4573.64
Bing's House	5	2.18E-7	1.04	3.15	34.55	378.84	4153.90
Heart	6	6.84E-8	1.13	2.26	30.55	412.15	5559.76
Brain	6	3.13E-8	1.20	1.97	31.30	496.07	7862.20
2-Complex	9	6.78E-8	1.24	6.18	107.46	1867.37	32451.12
3-Complex	9	7.89E-9	1.36	2.17	49.78	1140.45	26126.25
4-Complex	9	2.82E-9	1.45	1.78	50.15	1413.35	39833.56
5-Complex	9	4.95E-9	1.38	1.64	39.32	943.20	22625.87
6-Complex	9	3.04E-9	1.41	1.33	34.11	876.75	22535.83
7-Complex	9	2.42E-8	1.23	2.01	34.18	580.52	9858.60
8-Complex	9	2.26E-9	1.43	1.19	31.92	859.23	23126.42

Table 3. (a) Betti numbers of the files in the heart and brain datasets. (b) Number of reduction calls per dataset.

File Name	Number of Generators
heart192	(11, 9, 0)
heart160	(4, 15, 2)
heart128	(39, 12, 6)
heart96	(30, 107, 19)
heart64	(38, 228, 104)
heart32	(111, 220, 208)
brain192	(11, 0, 0)
brain160	(109, 1, 0)
brain128	(230, 25, 0)
brain96	(255, 153, 4)
brain64	(134, 407, 21)
brain32	(41, 524, 252)

(a)

Dataset	N	Reduction Calls		
		Avg	Max	SD
Torus	5	1.71	2.20	0.46
Heart	6	3.62	4.33	0.33
Brain	6	3.18	3.33	0.20
2-Complex	9	2.76	3.33	0.34
3-Complex	9	2.68	3	0.27
4-Complex	9	2.94	3.56	0.35
5-Complex	9	2.51	2.78	0.19
6-Complex	9	2.55	2.89	0.21
7-Complex	9	2.89	3.22	0.24
8-Complex	9	2.61	3	0.28
All others	N/A	1	1	0

(b)

parameters α and β for the heart and brain datasets, remain in the same order as those of d-balls and d-spheres despite their high Betti numbers as shown in Table 3(a).

For these two datasets, the high Betti numbers were not a relevant factor for the performance of the algorithm. We think it is because they contain a high ratio of exterior face reductions, which cost near to nothing. If we compare with the random d-complexes then the topology is an important factor. Indeed, as the data size increased we observed bigger differences in the time performance. Another interesting aspect of the algorithm is the number of recursions which is expressed as the number of reduction calls in Table 3(b).

6 Conclusion

Our experimentations show that this algorithm performs significantly faster than the classical reduction method. In addition, for all the datasets that we tested, which cover a wide range of types of data, its global performance indicated a subquadratic time complexity. Moreover, it allows to calculate the homology generators at a small additional time cost. Interestingly enough, in all the datasets we dealt with, the algorithm needed only to construct few intermediate complexes to obtain the homology of the original complex.

References

1. Allili, M., Corriveau, D., Ziou, D.: Morse Homology Descriptor for Shape Characterization. Proceedings of the 17th ICPR 4, 27–30 (2004)
2. Allili, M., Corriveau, D.: Topological Analysis of Shapes using Morse Theory. Computer Vision and Image Understanding 105, 188–199 (2007)
3. Collins, A., Zomorodian, A., Carlsson, G., Guibas, L.: A Barcode Shape Descriptor for Curve Point Cloud Data. Computers and Graphics 28, 881–894 (2004)
4. Munkres, J.R.: Elements of Algebraic Topology. Addison-Wesley, Reading (1984)
5. Kaczynski, T., Mischaikow, K., Mrozek, M.: Computational Homology. Appl. Math. Sci. Series, vol. 157. Springer, New York (2004)
6. Storjohann, A.: Near Optimal Algorithms for Computing Smith Normal Forms of Integer Matrices. In: Proceedings of 1996 International Symposium on Symbolic and Algebraic Computation, ISSAC 1996, pp. 267–274 (1996)
7. Peltier, S., Alayrangues, S., Fuchs, L., Lachaud, J.: Computation of Homology Groups and Generators. Computers and Graphics 30, 62–69 (2006)
8. Kaczynski, T., Mrozek, M., Slusarek, M.: Homology Computation by Reduction of Chain Complexes. Computers and Mathematics with App. 35, 59–70 (1998)
9. Mrozek, M., Pilarczyk, P., Zelazna, N.: Homology Algorithm Based on Acyclic Subspace. Computers and Mathematics with App. 55(11), 2395–2412 (2008)
10. Mrozek, M., Batko, B.: Coreduction Homology Algorithm. Discrete and Computational Geometry (in press)
11. Allili, M., Corriveau, D., Derivière, S., Kaczynski, T., Trahan, A.: Discrete Dynamical System Framework for Construction of Connections between Critical Regions in Lattice Height Data. J. Math. Imaging Vis. 28(2), 99–111 (2007)